Sparse complete sets for coNP: Solution of the P versus NP problem

Frank Vega
Joysonic, Uzun Mirkova 5, Belgrade, 11000, Serbia
vega.frank@gmail.com
https://orcid.org/0000-0001-8210-4126

Abstract

P versus NP is considered as one of the most important open problems in computer science. This consists in knowing the answer of the following question: Is P equal to NP? A precise statement of the P versus NP problem was introduced independently in 1971 by Stephen Cook and Leonid Levin. Since that date, all efforts to find a proof for this problem have failed. Another major complexity class is coNP. Whether NP = coNP is another fundamental question that it is as important as it is unresolved. In 1979, Fortune showed that if any sparse language is coNP-complete, then P = NP. We prove there is a possible sparse language in coNP-complete. In this way, we demonstrate the complexity class P is equal to NP.

2012 ACM Subject Classification Theory of computation → Complexity classes, Theory of computation → Problems, reductions and completeness

Keywords and phrases Complexity Classes, Sparse, Complement Language, Completeness, Polynomial Time

1 Issues

The P versus NP problem is a major unsolved problem in computer science [6]. This is considered by many to be the most important open problem in the field [6]. It is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute to carry a US$1,000,000 prize for the first correct solution [6]. It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency [1]. However, the precise statement of the $P = NP$ problem was introduced in 1971 by Stephen Cook in a seminal paper [6].

In 1936, Turing developed his theoretical computational model [18]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [18]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [18]. A nondeterministic Turing machine could contain more than one action defined for each step of its program, where this one is no longer a function, but a relation [18].

Another relevant advance in the last century has been the definition of a complexity class. A language over an alphabet is any set of strings made up of symbols from that alphabet [7]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [7].

In the computational complexity theory, the class P contains those languages that can be decided in polynomial time by a deterministic Turing machine [12]. The class NP consists in those languages that can be decided in polynomial time by a nondeterministic Turing machine [12]. Whether $P = NP$ or not is still a controversial and unsolved problem [1]. In this work, we proved the complexity class P is equal to NP. Hence, we solved one of the most important open problems in computer science.

2 Motivation

The biggest open question in theoretical computer science concerns the relationship between these classes: Is P equal to NP? In 2012, a poll of 151 researchers showed that 126 (83%) believed the answer to be no, 12 (9%) believed the answer is yes, 5 (3%) believed the question may be independent of the currently accepted axioms and therefore impossible to prove or disprove, 8 (5%) said either do not know or do not care or don't want the answer to be yes nor the problem to be resolved [11]. It is fully expected that $P \neq NP$ [17]. Indeed, if $P = NP$ then there are stunning practical consequences [17]. For that reason, $P = NP$ is considered as a very unlikely event [17]. Certainly, P versus NP is one of the greatest open problems in science and a correct solution for this incognita will have a great impact not only for computer science, but for many other fields as well [1].

3 Summary

In computational complexity theory, a sparse language is a formal language (a set of strings) such that the complexity function, counting the number of strings of length n in the language, is bounded by a polynomial function of n. The complexity class of all sparse languages is called $SPARSE$. $SPARSE$ contains $TALLY$, the class of unary languages, since these have at most one string of any one length.

Fortune showed in 1979 that if any sparse language is *coNP–complete*, then $P = NP$ (this is Fortune's theorem) [9]. Mahaney used this to show in 1982 that if any sparse language is *NP–complete*, then $P = NP$ [14]. A simpler proof of this based on left-sets was given by Ogihara and Watanabe in 1991 [16]. Mahaney's argument does not actually require the sparse language to be in NP, so there is a sparse *NP–hard* set if and only if $P = NP$ [14].

We create a class with the opposite definition, that is a class of languages that are dense instead of sparse. We show there is a sequence of languages that are in *NP–complete*, but their density grows as much as we go forward into the iteration of the sequence. The first element of the sequence is a variation of the *NP–complete* problem known as *HAM–CYCLE* [10]. The next element in the sequence is constructed from this new version of *HAM–CYCLE*. Indeed, each language is created from its previous language in the sequence.

Since the density grows according we move forward into the sequence, then there must be a language so much dense such that its complement is sparse. Fortunately, we find this property from a language of this sequence when the bit length n of the binary strings tends to infinity. However, this incredible dense language is still *NP–complete*. Thus, the complement of this language remains in *coNP–complete*, because the complement of every *NP–complete* language is complete for $coNP$ [12].

In this way, we find a sparse language in *coNP–complete*. As a consequence of Fortune's theorem, we demonstrate that P is equal to NP. To sum up, we proved there is a sparse complete set for $coNP$ and therefore, we just solved the P versus NP problem.

4 Significance

No one has been able to find a polynomial time algorithm for any of more than 300 important known *NP–complete* problems [10]. A proof of $P = NP$ will have stunning practical consequences, because it leads to efficient methods for solving some of the important problems in NP [6]. The consequences, both positive and negative, arise since various *NP–complete* problems are fundamental in many fields [6]. This result explicitly concludes supporting the existence of a practical solution for the *NP–complete* problems because $P = NP$.

Cryptography, for example, relies on certain problems being difficult. A constructive and efficient solution to an *NP–complete* problem such as $3SAT$ will break most existing cryptosystems including: Public-key cryptography [13], symmetric ciphers [15] and one-way functions used in cryptographic hashing [8]. These would need to be modified or replaced by information-theoretically secure solutions not inherently based on *P–NP* equivalence.

There are enormous positive consequences that will follow from rendering tractable many currently mathematically intractable problems. For instance, many problems in operations research are *NP–complete*, such as some types of integer programming and the traveling salesman problem [10]. Efficient solutions to these problems have enormous implications for logistics [6]. Many other important problems, such as some problems in protein structure prediction, are also *NP–complete*, so this will spur considerable advances in biology [3].

But such changes may pale in significance compared to the revolution an efficient method for solving *NP–complete* problems will cause in mathematics itself. Research mathematicians spend their careers trying to prove theorems, and some proofs have taken decades or even centuries to find after problems have been stated. For instance, Fermat's Last Theorem took over three centuries to prove. A method that is guaranteed to find proofs to theorems, should one exist of a "reasonable" size, would essentially end this struggle.

Besides, a $P = NP$ proof reveals the existence of an interesting relationship between humans and machines [1]. For example, suppose we want to program a computer to create new Mozart-quality symphonies and Shakespeare-quality plays. When $P = NP$, this could be reduced to the easier problem of writing a computer program to recognize great works of art [1].

5 Basic Definitions

Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of finite strings over Σ [2]. A Turing machine M has an associated input alphabet Σ [2]. For each string w in Σ^* there is a computation associated with M on input w [2]. We say that M accepts w if this computation terminates in the accepting state, that is $M(w) =$ "yes" [2]. Note that M fails to accept w either if this computation ends in the rejecting state, that is $M(w) =$ "no", or if the computation fails to terminate [2].

The language accepted by a Turing machine M, denoted $L(M)$, has an associated alphabet Σ and is defined by

$$L(M) = \{w \in \Sigma^* : M(w) = \text{“}yes\text{”}\}.$$

We denote by $t_M(w)$ the number of steps in the computation of M on input w [2]. For $n \in \mathbb{N}$ we denote by $T_M(n)$ the worst case run time of M; that is

$$T_M(n) = max\{t_M(w) : w \in \Sigma^n\}$$

where Σ^n is the set of all strings over Σ of length n [2]. We say that M runs in polynomial time if there is a constant k such that for all n, $T_M(n) \leq n^k + k$ [2]. In other words, this means the language $L(M)$ can be accepted by the Turing machine M in polynomial time. Therefore, P is the complexity class of languages that can be accepted in polynomial time by deterministic Turing machines [7]. A verifier for a language L is a deterministic Turing machine M, where

$$L = \{w : M(w, c) = \text{“}yes\text{”} \textit{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w, so a polynomial time verifier runs in polynomial time in the length of w [2]. A verifier uses additional information, represented by the symbol c, to verify that a string w is a member of L. This information is called certificate. NP is also the complexity class of languages defined by polynomial time verifiers [17]. If NP is the class of problems that have succinct certificates, then the complexity class $coNP$ must contain those problems that have succinct disqualifications [17]. That is, a "*no*" instance of a problem in $coNP$ possesses a short proof of its being a "*no*" instance [17].

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some deterministic Turing machine M, on every input w, halts in polynomial time with just $f(w)$ on its tape [18]. Let $\{0,1\}^*$ be the infinite set of binary strings, we say that a language $L_1 \subseteq \{0,1\}^*$ is polynomial time reducible to a language $L_2 \subseteq \{0,1\}^*$, written $L_1 \leq_p L_2$, if there is a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$x \in L_1$ *if and only if* $f(x) \in L_2$.

An important complexity class is *NP–complete* [12]. A language $L \subseteq \{0,1\}^*$ is *NP–complete* if

- $L \in NP$, and
- $L' \leq_p L$ for every $L' \in NP$.

If L is a language such that $L' \leq_p L$ for some $L' \in$ *NP–complete*, then L is *NP–hard* [12]. Moreover, if $L \in NP$, then $L \in$ *NP–complete* [12]. A principal *NP–complete* problem is *HAM–CYCLE* [7].

An instance of the language *HAM–CYCLE* is a simple graph $G = (V, E)$ where V is the set of vertices and E is the set of edges, each edge being an unordered pair of vertices [7]. We say $(u, v) \in E$ is an edge in a simple graph $G = (V, E)$ where u and v are vertices. A simple graph is an undirected graph without multiple edges or loops [7]. For a simple graph $G = (V, E)$ a simple cycle in G is a sequence of distinct vertices $\langle v_0, v_1, v_2, ..., v_k \rangle$ such that $(v_k, v_0) \in E$ and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, ..., k$ [7]. A Hamiltonian cycle is a simple cycle of the simple graph which contains all the vertices of the graph. A simple graph that contains a hamiltonian cycle is said to be hamiltonian; otherwise, it is nonhamiltonian [7]. The problem *HAM–CYCLE* asks whether a simple graph is hamiltonian [7].

6 Results

▶ **Definition 1.** A dense language on m is a formal language (a set of **binary** strings) such that for a positive integer n_0, the counting of the number of strings of length $n \geq n_0$ in the language is greater than or equal to 2^{n-m} where m is a real number and $0 \leq m \leq 1$. The complexity class of all dense languages on m is called $DENSE(m)$.

In this work, we are going to represent the simple graphs with an adjacency-matrix [7]. For the adjacency-matrix representation of a simple graph $G = (V, E)$, we assume that the vertices are numbered $1, 2, \ldots, |V|$ in some arbitrary manner. The adjacency-matrix representation of a simple graph G consists of a $|V| \times |V|$ matrix $A = (a_{i,j})$ such that $a_{i,j} = 1$ when $(i, j) \in E$ and $a_{i,j} = 0$ otherwise [7]. In this way, every simple graph of k vertices is represented by k^2 bits.

Observe the symmetry along the main diagonal of the adjacency matrix in this kind of graph that is called simple. We define the transpose of a matrix $A = (a_{i,j})$ to be the matrix

$A^T = (a^T_{i,j})$ given by $a^T_{i,j} = a_{j,i}$. Hence the adjacency matrix A of a simple graph is its own transpose $A = A^T$.

▶ **Definition 2.** The language *NON–SIMPLE* contains all the graph that are represented by an adjacency-matrix A such that $A \neq A^T$

▶ **Lemma 3.** *NON–SIMPLE* $\in P$.

Proof. Given a binary string x, we can check whether x is an adjacency-matrix which is not equal to its own transpose in time $O(|x|^2)$ just iterating each bit $a_{i,j}$ in x and checking whether $a_{i,j} \neq a_{j,i}$ or not where $|\dots|$ represents the bit-length function [7]. ◀

▶ **Definition 4.** The language *HAM–CYCLE'* contains all the binary strings z such that $z = xy$, the bit-length of x is equal to $(\lfloor\sqrt{|z|}\rfloor)^2$ and $x \in$ *HAM–CYCLE* or $x \in$ *NON–SIMPLE* where $|\dots|$ represents the bit-length function and y could be the empty string.

▶ **Lemma 5.** *HAM–CYCLE'* $\in$ *NP–complete.*

Proof. Given a binary string x we can decide in polynomial time whether $x \notin$ *NON–SIMPLE* just verifying when $x = x^T$. In this way, we can reduce in polynomial time a simple graph $G = (V, E)$ of k vertices encoded as the binary string x such that when x has k^2 bits and $x \notin$ *NON–SIMPLE* then

$x \in$ *HAM–CYCLE if and only if* $x \in$ *HAM–CYCLE'*.

Then, we can reduce in polynomial time each element of *HAM–CYCLE* to *HAM–CYCLE'*. Therefore, *HAM–CYCLE'* is in *NP–hard*. Moreover, we can check in polynomial time whether a binary string z such that $z = xy$ where the bit-length of x is equal to $(\lfloor\sqrt{|z|}\rfloor)^2$ and complies with $x \in$ *HAM–CYCLE* or $x \in$ *NON–SIMPLE* since *HAM–CYCLE* $\in NP$, *NON–SIMPLE* $\in P$ and $P \subseteq NP$ [17]. Consequently, *HAM–CYCLE'* is in *NP*. Hence, *HAM–CYCLE'* $\in$ *NP–complete*. ◀

▶ **Lemma 6.** *HAM–CYCLE'* $\in DENSE(1)$.

Proof. *OEIS A000088* gives the total number of graphs on n unlabeled points [19]. For 8 points there are 12346 so just over half the graphs on 8 points are Hamiltonian [19]. For 12 points, the highest in the Hamiltonian list, there are 152522187830 Hamiltonian graphs out of 165091172592 which would claim that over 92% of the 12 point graphs are Hamiltonian [19]. For $n = 2$ there are two graphs, neither of which is Hamiltonian [19]. For $n < 8$ over half the graphs are not Hamiltonian [19]. It does not seem surprising that once n gets large most graphs are Hamiltonian [19].

Choosing a graph on n vertices at random is the same as including each edge in the graph with probability $\frac{1}{2}$, independently of the other edges [4]. You get a more general model of random graphs if you choose each edge with probability p [4]. This model is known as $G_{n,p}$ [4]. It turns out that for any constant $p > 0$, the probability that G contains a Hamiltonian cycle tends to 1 when n tends to infinity [4]. In fact, this is true whenever $p > \frac{c \times \log n}{n}$ for some constant c. In particular this is true for $p = \frac{1}{2}$, which is our case [4].

For all the binary strings z such that $z = xy$ where the bit-length of x is equal to $(\lfloor\sqrt{|z|}\rfloor)^2$, the amount of elements of size $|z|$ in *HAM–CYCLE'* is equal to the number of binary strings $x \in$ *HAM–CYCLE* or $x \in$ *NON–SIMPLE* multiplied by $2^{|z|-(\lfloor\sqrt{|z|}\rfloor)^2}$. Since the number of Hamiltonian graphs increases as much as we go further on n, it does not seem surprising either that once n gets large most binary strings belong to *HAM–CYCLE'*.

Certainly, we can affirm for a sufficiently large positive integer n'_0, all the binary strings of length $n \geq n'_0$ which belong to *HAM–CYCLE'* are indeed more than or equal to 2^{n-1} elements. In this way, we prove $HAM\text{–}CYCLE' \in DENSE(1)$. ◀

▶ **Definition 7.** We will define a sequence of languages $HAM\text{–}CYCLE'_k$ for every possible integer $1 \leq k$. We state $HAM\text{–}CYCLE'_1$ as the language *HAM–CYCLE'*. Recursively, from a language $HAM\text{–}CYCLE'_k$, we define $HAM\text{–}CYCLE'_{k+1}$ as follows: A binary string xy complies with $xy \in HAM\text{–}CYCLE'_{k+1}$ if and only if x and y are binary strings, $x \in HAM\text{–}CYCLE'_k$ or $y \in HAM\text{–}CYCLE'_k$ such that $|x| = |y|$ when $|xy|$ is even and $|x|+1 = |y|$ when $|xy|$ is odd where $|\dots|$ represents the bit-length function.

▶ **Lemma 8.** *For every integer* $1 \leq k$, $HAM\text{–}CYCLE'_k \in NP$.

Proof. This is true for $k = 1$. Every string xy which belongs to $HAM\text{–}CYCLE'_2$ complies with $x \in HAM\text{–}CYCLE'_1$ or $y \in HAM\text{–}CYCLE'_1$ such that $|x| = |y|$ when $|xy|$ is even and $|x|+1 = |y|$ when $|xy|$ is odd. Moreover, every string $xyvw$ which belongs to $HAM\text{–}CYCLE'_3$ complies with $x \in HAM\text{–}CYCLE'_1$ or $y \in HAM\text{–}CYCLE'_1$ or $v \in HAM\text{–}CYCLE'_1$ or $w \in HAM\text{–}CYCLE'_1$ such that $|xy| = |vw|$ when $|xyvw|$ is even and $|xy| + 1 = |vw|$ when $|xyvw|$ is odd, $|x| = |y|$ when $|xy|$ is even and $|x| + 1 = |y|$ when $|xy|$ is odd and $|v| = |w|$ when $|vw|$ is even and $|v| + 1 = |w|$ when $|vw|$ is odd. Furthermore, we can extend this property for every positive integer $k > 3$ in $HAM\text{–}CYCLE'_k$. Indeed, $HAM\text{–}CYCLE'_k$ is in NP for every integer $1 \leq k$, because the verification of whether the whole string or substrings are indeed elements of $HAM\text{–}CYCLE'_1$ can be done in polynomial time with the appropriated certificates. ◀

▶ **Theorem 9.** *For every integer* $1 \leq k$, $HAM\text{–}CYCLE'_k \in NP\text{–}complete$.

Proof. This is true for $k = 1$ by Lemma 5. Let's assume is valid for some positive integer $1 \leq k'$. Let's prove this for $k' + 1$. We already know the adjacency-matrix of n^2 zeros represents a simple graph of n vertices which does not contain any edge. This kind of a simple graph does not belong to $HAM\text{–}CYCLE'_1$. Suppose, we have an instance y of $HAM\text{–}CYCLE'_{k'}$. We can reduce y in $HAM\text{–}CYCLE'_{k'}$ to zy in $HAM\text{–}CYCLE'_{k'+1}$ such that

$$y \in HAM\text{–}CYCLE'_{k'} \text{ if and only if } zy \in HAM\text{–}CYCLE'_{k'+1}$$

where the binary string z is exactly a sequence of $|y|$ zeros. Due to this reduction remains in polynomial time for every positive integer $1 \leq k'$, then we show $HAM\text{–}CYCLE'_{k'+1}$ is in *NP–hard*. Moreover, $HAM\text{–}CYCLE'_{k'+1}$ is also in *NP–complete*, because of Lemma 8. ◀

▶ **Theorem 10.** *For every integer* $1 \leq k$, *if the language* $HAM\text{–}CYCLE'_k$ *is in* $DENSE(k')$ *for every natural number* $n' \geq n_0$, *then* $HAM\text{–}CYCLE'_{k+1}$ *is in* $DENSE(\frac{k'}{2})$ *for every integer* $n' \geq 2 \times n_0 + 1$.

Proof. If the language $HAM\text{–}CYCLE'_k$ is in $DENSE(k')$ for every natural number $n' \geq n_0$, then for every integer $n \geq n_0 + 1$ the amount of elements of size $n + i$ in $HAM\text{–}CYCLE'_{k+1}$ (where $i = n$ or $i = n - 1$) is greater than or equal to

$$2^{i-k'} \times 2^n + 2^{n-k'} \times (2^i - 2^{i-k'}).$$

This is because there must be more than or equal to $2^{i-k'}$ elements of size i in $HAM\text{–}CYCLE'_k$ which are prefixes of the binary strings of size $n + i$ in the language $HAM\text{–}CYCLE'_{k+1}$.

Moreover, there must be more than or equal to $2^{n-k'}$ elements of size n in *HAM–CYCLE'*$_k$ which are suffixes of the binary strings of size $n+i$ in *HAM–CYCLE'*$_{k+1}$. If we join both properties, we obtain the sum described by the formula above.

Indeed, this formula can be simplified to

$$2^{n+i-k'} + 2^{n+i-k'} \times (2^0 - 2^{-k'})$$

and extracting a common factor we obtain

$$2^{n+i-k'} \times (1 + (1 - 2^{-k'})$$

which is equal to

$$2^{n+i-k'} \times (2 - \frac{1}{2^{k'}}).$$

Nevertheless, for every real number $0 \leq k' \leq 1$

$$(2 - \frac{1}{2^{k'}}) \geq 2^{\frac{k'}{2}}.$$

Certainly, if we multiply both member of the inequality by $2^{k'}$, we obtain

$$(2^{k'+1} - 1) \geq 2^{k'+\frac{k'}{2}}$$

which is equivalent to

$$2^{k'} \times (2 - 2^{\frac{k'}{2}}) \geq 1$$

that it is true for every real number $0 \leq k' \leq 1$. Thus

$$2^{n+i-k'} \times (2 - \frac{1}{2^{k'}}) \geq 2^{n+i-k'} \times 2^{\frac{k'}{2}}$$

where

$$2^{n+i-k'} \times 2^{\frac{k'}{2}} = 2^{n+i-(k'-\frac{k'}{2})} = 2^{n+i-\frac{k'}{2}}.$$

Since every binary string of size n' has also the bit-length $n+i$ for some natural number n (where $i = n$ or $i = n-1$), then there are more than or equal to $2^{n'-(\frac{k'}{2})}$ elements of the language *HAM–CYCLE'*$_{k+1}$ with length $n' \geq 2 \times n_0 + 1$. In this way, we show *HAM–CYCLE'*$_{k+1}$ is in $DENSE(\frac{k'}{2})$ for every integer $n' \geq 2 \times n_0 + 1$. ◀

▶ **Lemma 11.** *HAM–CYCLE'*$_k \in DENSE(\frac{1}{2^{k-1}})$ *for every natural number* $n \geq 2^{k-1} \times n'_0 + 2^{k-1} - 1$ *where the constant* n'_0 *is the positive integer used in the Definition 1 and Lemma 6 for HAM–CYCLE'.*

Proof. According to Lemma 6, *HAM–CYCLE'*$_1$ is in $DENSE(1)$ for every natural number $n \geq n'_0 = 2^{1-1} \times n'_0 + 2^{1-1} - 1$. Consequently, due to Theorem 10, *HAM–CYCLE'*$_2$ is in $DENSE(\frac{1}{2})$ for every natural number $n \geq 2 \times n'_0 + 1 = 2^{2-1} \times n'_0 + 2^{2-1} - 1$. Moreover, *HAM–CYCLE'*$_3$ is in $DENSE(\frac{1}{4})$ for every natural number $n \geq 4 \times n'_0 + 3 = 2^{3-1} \times n'_0 + 2^{3-1} - 1$ and so forth ... and thus, for every language *HAM–CYCLE'*$_k$, we have *HAM–CYCLE'*$_k \in DENSE(\frac{1}{2^{k-1}})$ for every natural number $n \geq 2^{k-1} \times n'_0 + 2^{k-1} - 1$. ◀

▶ **Corollary 12.** *There is some language HAM–CYCLE'*$_k$ *such that HAM–CYCLE'*$_k \in DENSE(0)$ *when the bit length* n *of the binary strings tends to infinity.*

Proof. When k tends to infinity, then $\frac{1}{2^{k-1}}$ tends to 0. In this way, when k tends to infinity, then $HAM\text{–}CYCLE'_k \in DENSE(0)$ as a consequence of Lemma 11. However, when k grows, then the constant n_0 becomes exponentially larger in relation to k where n_0 is the positive integer used in the Definition 1 for $HAM\text{–}CYCLE'_k$. In this way, the density totally grows for some language $HAM\text{–}CYCLE'_k$ when the bit length n of the binary strings tends to infinity. Consequently, this language $HAM\text{–}CYCLE'_k$ may actually exist. ◀

▶ **Theorem 13.** *There is a sparse language in coNP–complete.*

Proof. In Corollary 12, the complement of some language $HAMILTON\text{–}PATH'_k$ is sparse when the bit length n of the binary strings tends to infinity. Thus, the complexity of counting the number of strings with length n in the complement of this language is bounded by a polynomial function of n. Indeed, a language is sparse if and only if its complement is in $DENSE(0)$ when the bit length n of the binary strings tends to infinity [14]. Indeed, the sparse languages are called sparse because there are a total of 2^n strings of length n, and if a language only contains polynomially many of these, then the proportion of strings of length n that it contains rapidly goes to zero as n grows (which means its complement should be in $DENSE(0)$ when n tends to infinity) [14]. However, according to Theorem 9, the complements of these languages $HAMILTON\text{–}PATH'_k$ must be in *coNP-complete*, because the complements of the *NP-complete* problems are complete for $coNP$. ◀

▶ **Lemma 14.** $P = NP$.

Proof. By the Fortune's theorem, if any sparse language is *coNP–complete*, then $P = NP$ [9]. As result of Theorem 13, there is a sparse language in *coNP–complete*. Finally, we demonstrate that P is equal to NP. ◀

7 Discussion

A logarithmic space Turing machine has a read-only input tape, a write-only output tape, and a read/write work tape [18]. The work tape may contain $O(\log n)$ symbols [18]. In computational complexity theory, $LOGSPACE$ is the complexity class containing those decision problems that can be decided by a logarithmic space Turing machine which is deterministic [17]. Whether $LOGSPACE = P$ is another fundamental question that it is as important as it is unresolved [17].

A logarithmic space Turing machine M may compute a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the string remaining on the output tape after M halts when it is started with w on its input tape [18]. We call f a logarithmic space computable function [18]. We say that a language $L_1 \subseteq \{0,1\}^*$ is logarithmic space reducible to a language $L_2 \subseteq \{0,1\}^*$, written $L_1 \leq_l L_2$, if there exists a logarithmic space computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$x \in L_1$ *if and only if* $f(x) \in L_2$.

The logarithmic space reduction is frequently used for the class *P-complete* [17].

In 1999, Jin-Yi Cai and D. Sivakumar, building on work by Ogihara, showed that if there exists a sparse *P-complete* problem, then $LOGSPACE = P$ [5]. We might extend the proof of this paper to demonstrate that $LOGSPACE = P$. Certainly, we might only need to find some *P-complete* which belongs to $DENSE(1)$ because the *P-completeness* is closed under complement [17]. Indeed, the other steps of that possible proof might be similar to the arguments that we follow in this paper. Consequently, this work would help us not only to solve P versus NP, but also $LOGSPACE$ versus P.

References

1. Scott Aaronson. P $\stackrel{?}{=}$ NP. *Electronic Colloquium on Computational Complexity, Report No. 4*, 2017.
2. Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
3. Bonnie Berger and Tom Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.
4. Béla Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 2001. doi:10.1017/CBO9780511814068.
5. Jin-Yi Cai and D. Sivakumar. Sparse hard sets for P: resolution of a conjecture of Hartmanis. *Journal of Computer and System Sciences*, 58(2):280–296, 1999.
6. Stephen A Cook. The P versus NP Problem, April 2000. at http://www.claymath.org/sites/default/files/pvsnp.pdf.
7. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
8. Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion attacks on secure hash functions using SAT solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 377–382. Springer, 2007.
9. S. Fortune. A note on sparse complete sets. *SIAM Journal on Computing*, 8(3):431–433, 1979.
10. Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company, 1 edition, 1979.
11. William I Gasarch. Guest column: The second P $\stackrel{?}{=}$ NP poll. *ACM SIGACT News*, 43(2):53–77, 2012.
12. Oded Goldreich. *P, NP, and NP-Completeness: The basics of computational complexity*. Cambridge University Press, 2010.
13. Satoshi Horie and Osamu Watanabe. Hard instance generation for SAT. *Algorithms and Computation*, pages 22–31, 1997.
14. S. R. Mahaney. Sparse complete sets for NP: Solution of a conjecture by Berman and Hartmanis. *Journal of Computer and System Sciences*, 25:130–143, 1982.
15. Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24(1):165–203, 2000.
16. M. Ogiwara and O. Watanabe. On polynomial time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20:471–483, 1991.
17. Christos H Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
18. Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.
19. The On-Line Encyclopedia of Integer Sequences. Number of graphs on n unlabeled nodes, August 2018. at http://oeis.org/A000088.